Assumptions made when designing/implementing the API:

- Each entry should contain a name -> Instead of having two variables for first and last name, we use a name variable which contains the first name, followed by one or more white space characters and the last name. Using a regular expression, we only allow the variable name to get values based the format explained above. If we need to access either the first or the last name of any listing, we can use split () for whitespace, which will return a string array with first and last names in it.

- Each entry should contain a type -> If we were to save the type into a string variable, we would continuously save the same word over and over again. Thus, we use an integer variable which can only get values ranging from 0 to 2, for each corresponding type, to save space. Each integer takes up about 4 bytes whereas a string can take up to a byte per letter.

- Each entry should contain a number-> A phone number has 10 digits. The biggest number we can save in an integer variable is 4.294.967.295 and in a long variable is 18.446.744.073.709.551.615. Even though, a long variable would be able to save a phone number in less bytes (8 bytes) than a string of 10 character (10 bytes), if we want to take the phonebook to a global scale, adding country code (e.g. +30, +1, …) or any operations done on the number, would be easier with a string variable.

- The file format should not be text or DB based -> For each different listing of the phone book we create an object of the class “listing” which contains it`s information (name, type, and number). We create a list containing all the listings and we serialize it to a binary format. If we want to access the data, we simply deserialize the list, we perform the operations we want on it, and then we serialize it again saving the changes we made. This also makes our data portable, since all that a UI would need to display or edit them, would be the stored file and a reference to our phonebook_lib.

- I have created a simple windows form interface to delete, edit, add, and sort data using the reusable phonebook library, for basic debugging purposes.